U.S. DEPARTMENT OF COMMERCE
Patent and Trademark Offic

148

DOCUMENT RETRIEVAL REQUEST FORM

Please include RightFax Number to expedite return of documents

Requester's Name: Lucas Divine	Case Serial Number: 09/709486		Art Unit/Org.: 2624
Phone: 27432	**RightFax:	Building: Knox	Room Number: 9D25
Class/Sub-Class:			
Date of Request: 7/21/05		Date Needed By: 7/22/05	

Paste or add text of citation or bibliography: **Paste Citation** Only one request per form. Original copy only. ☐

Author/Editor:	
Journal/Book Title:	
Article Title:	
Volume Number:	Report Number: Pages:
Issue Number:	Series Number: Year of Publication:
Publisher:	
Remarks:	see attached

Staff Use Only

Monthly Accession Number:

Library Action	PTO		LC		NAL		NIH		NLM		NIST		Other	
	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd
Local Attempts														
Date	7/21													
Initials	KEJ													
Results	Comp													
Examiner Called														
Page Count														
Money Spent														

	Source	Date
Ordered From:		

Remarks/Comments
1st and 2nd denotes time taken to a library

O/N – Under NLM means Overnight

Comments:

5879259 INSPEC Abstract Number: C9805-6160B-022
Title: **Database access with Java and** Active Server Pages
Author(s): Casale, A.
Journal: WEB Techniques vol.3, no.4 p.55-8
Publisher: Miller Freeman,
Publication Date: April 1998 Country of Publication: USA
CODEN: WETEFA ISSN: 1086-556X
SICI: 1086-556X(199804)3:4L.55:DAWJ;1-P
Material Identity Number: F184-98003
Language: English
Subfile: C

Title: Database access with Java and Active Server Pages
...Abstract: combine them is quite another. In this article, I show how to combine HTML, Java, **Active Server Pages** and Access 97 to create a World Wide Web database application. I share my experiences...
...Identifiers: **Active Server Pages** ; ...

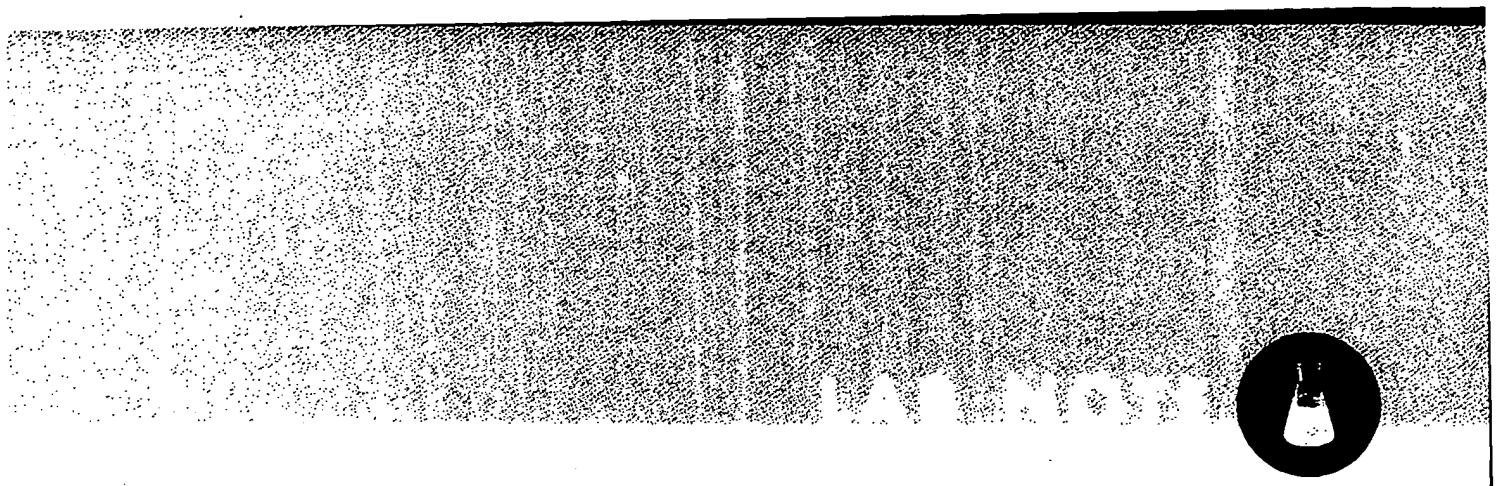
... **Microsoft** Access 97

13/3,K/21 (Item 21 from file: 2)
DIALOG(R)File 2:INSPEC

5771543 INSPEC Abstract Number: C9801-6115-023
Title: Keeping active. A view of Microsoft **'s Active Platform and Visual InterDev**
Author(s): Spitzer, T.
Author Affiliation: EC Co., Silicon Valley, CA, USA
Journal: DBMS vol.10, no.13 p.75-6, 78-9
Publisher: Miller Freeman,
Publication Date: Dec. 1997 Country of Publication: USA
CODEN: DBMSEO ISSN: 1041-5173
SICI: 1041-5173(199712)10:13L.75:KAVM;1-X
Material Identity Number: M772-97013
Language: English
Subfile: C

Title: Keeping active. A view of Microsoft **'s Active Platform and Visual InterDev**
...Abstract: more impact on developing intranet applications as your choice of client does. I've chosen **Microsoft** 's Internet Information Server (IIS), and my team is getting geared up to build registration... ... well as to internal customer service personnel. My current plan is to employ a "mostly **Microsoft** " solution for building these applications, using the **Microsoft** Visual InterDev site-development tool to coordinate active database object connections, HTML pages and **active server pages** . The principal component of our solution that's not from **Microsoft** is an Oracle database. We decided to use Oracle as the backbone for our electronic...

... our support applications as our production applications. In this article, I present an overview of **Microsoft** 's Active Platform as it stands today, share the thought process that led me to...
Identifiers: **Microsoft** Active Platform...



Database Access with Java and Active Server Pages

BY ARTHUR CASALE

I am continually astonished by the sheer number of technologies you deal with when developing Web database-applications. Just as amazing is the seemingly infinite number of ways these technologies can be knit together to form a solution. A case in point is a recent project that utilized HTML, Java, Microsoft's Active Server Pages (ASP) and the Access 97 database. The application was simple: Allow a Netscape Navigator or Internet Explorer user to issue the query "show me all employees in department XYZ and their current salaries" against an ODBC-compliant database on the server.

In this article, I'll share my experiences and show you how I assembled these various technologies to create this application. My objective is not to explore the capabilities and syntactic nuances of the technologies utilized, but to "baseline" several technologies and illustrate how they can be effectively integrated to form a seamless end-to-end, albeit no-frills, interactive-database solution. Each piece of the application can, of course, be expanded at your discretion to include additional functionality.

Application Architecture

The application's interaction allows a user to select a department number from a drop-down list in a Java applet and initiate a query by pressing a "Go" button. The Java applet passes the department entered to an ASP program sitting on the host server. The ASP code opens up a connection to an Access 97 database and passes an SQL query to the database for processing. The Access 97 engine processes the query and sends the result set (qualifying employees and their salaries) back to the ASP program. The ASP program then hands the result set to Java. Java parses through the result set and displays each employee/salary combination as a row in a text area of the applet.

Creating the Data Source

The Access database consists of just one table, "Employee," which is shown in Table 1. The Employee table contains the employee number (the primary key), and the employee's name (first and last), weekly salary, and department code. The first step is to ensure that the Access database is in your host directory on the server. In addition to storing the actual database, you must define a system-level data source name (DSN) on the server and associate the DSN with the physical location/name of the database. In this example, I've created an Access database called "empdept.mdb" and moved it to my host directory via FTP. I then asked my ISP to create a system DSN on the server and associate it with the "empdept.mdb" database. You can, of course, do this yourself—if you have rights or access to an NT server—using these steps:

1. From the START button on the Windows 95/NT desktop, go to Settings and then Control panel.
2. Click on the "32 ODBC" icon to begin the process of defining a System DSN.
3. Click on the "System DSN..." button.
4. Click on the "ADD" button.
5. Highlight "Miscrosoft Access Driver (*.mdb)" in the "Installed ODBC Driver" list and press the OK button.
7. Enter the System DSN (NTHOSTempdata) in the "Data Source Name" text box; then click on the "Select..." button to locate the Access database you want to associate with the DSN (C:\MYROOT\empdept.mdb). Click on OK.

LISTING ONE

```
<HTML>
<HEAD>
<TITLE>List Employees By Department</TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE="GetEmpsByDept.class" WIDTH=425 HEIGHT=225>
</APPLET>
</BODY>
</HTML>
```

Embedding the Applet

Both the processing and all the display elements (drop-down list, text area, and so on) are part of the Java applet. In this case, an HTML form (see Listing One) is simply used as a "container" for the applet. Figure 1 shows how the applet will appear in the browser. Listing One is straightforward: It simply invokes the "compiled" applet, GetEmpsByDept.class, and specifies the overall size in pixels of the display box that will be presented by the applet (`WIDTH` and `HEIGHT`).

Initialization

Listing Two contains the associated Java source listing. At the risk of oversimplifying, an applet can be broken into four logical pieces: initialization (setting variables and displaying the initial screen elements), starting an execution thread (sometimes), waiting for the user to do something (such as pressing a key), and responding to the user request. I'll simplify this further by skipping the thread process in this example.

The `init()` method in Listing Two starts by initializing required objects. The first order of business is to create the drop-down list of department codes that the user will select from. So, `init()` creates a "Choice" box containing the department codes. The `choice` object is instantiated with `choice = new Choice()` and then drop-down "items" are added to it. The `choice` object is added to the applet (making it available for display) using `add()`. We now have two more display elements that need to be added to the applet: a select button and a text area. (The user will click on the select button after selecting a department code.) The text area will be populated with the result set (employees and their salaries) that is returned by the ASP program. Next, I instantiate the select button (named `button` and labeled "Get Employee") and add the new `button` object to the applet. The `textarea` object is instantiated and added in the same way. Note that until they are added, the objects cannot be seen or manipulated by the user.

Event Processing

This is where it gets gnarly. After initialization, the applet just sits there and waits for the user to do something to one of its display elements. The `action()` method is invoked when an event occurs. In our case, the `action()` method is designed to poll or trap two sorts of user interaction: a selection from the drop-down department list or a click on the "Select Employees..." button. The two event types are associated with `if` statements in the `action()` method.

For the drop-down list, Listing Two moves the department code selected by the user to a `String` variable called `item`, which will be used in the "Select Employee" event process. The "Select Employees in..." button event logic is no more complicated, but triggers a complicated process. The `textarea.setText(getEmployees(item).toString()` statement invokes the `getEmployees()` method and passes it the department code (`item`), which was set in the event process for the drop-down list. The `getEmployees()` method returns a value that is then moved to the `textarea` display object using `setText()`. The value returned by the method is the actual list of employees and their salaries.

For its part, the `getEmployees()` method will return an object to the command line we are examining. Unfortunately, the object it returns is a `StringBuffer` object type and must be converted for display. That is the job of the `toString()` subcommand. The converted string must now be added to the `textarea` object; the `SetText` subcommand is used to effectively move that value to the `textarea` display object. Now our `textarea` object is populated with the employee/salary list, but since it was changed in this routine it

EMPLOYEE_NBR	FIRSTNAME	LASTNAME	SALARY	DEPARTMENT
0000000001	Brian	Basic	$400.00	001
0000000002	Kevin	Cobol	$500.65	001
0000000003	Paula	Perl	$648.55	003
0000000004	Cecil	Cgi	$892.52	003
0000000005	Jane	Java	$1,003.26	003
0000000006	Alvin	Activeserverpages	$856.94	003
0000000007	Astrid	Assembler	$123.35	003
0000000008	Linus	Linux	$489.66	003
0000000009	Jethro	Javascript	$847.44	003
0000000010	Paolo	Powerbuilder	$629.22	003

Table 1: Definition for the Employee table.

LISTING TWO

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.applet.Applet;
import java.net.URL;
import java.net.MalformedURLException;
public class GetEmpsByDept extends Applet {
    Choice choice; //pop-up list of choices
    Button button;
    TextArea textarea;
    TextArea textReturned;
    String department;
    public void init() {

        choice = new Choice();
        choice.addItem("C01");
        choice.addItem("C02");
        choice.addItem("C03");
        choice.addItem("C04");
        choice.addItem("C05");
        choice.addItem("C06");
        choice.addItem("C07");
        choice.addItem("C08");
        choice.addItem("C09");
        choice.addItem("C10");

     //Add components to the Applet.
        add(choice);
        button = new Button("   Select Employees in C01   ");
        add(button);
        textarea = new TextArea(8, 50);
        textarea.setEditable(false);
        add(textarea);
        validate();
        repaint();
    }
    public boolean action(Event e, Object arg) {
        if (e.target instanceof Choice) {
            int choiceidx = choice.getSelectedIndex();
            String item = choice.getItem(choiceidx);
            button.setLabel("Select Employees in " + item);
            return true;
        }
        if (e.target instanceof Button) {
            String item = choice.getSelectedItem();
            textarea.setText(" ");
            textarea.setText(getEmployees(item).toString());
            repaint();
            return true;
        }
      return false;
    }
    public void paint(Graphics g) {
        Dimension d = size();
        g.setColor(new Color(000,000,90));
        g.fill3DRect(0,0,d.width,d.height,true);
        g.draw3DRect(0,0,d.width - 2,d.height - 2,true);
        g.draw3DRect(3,3,d.width - 7,d.height - 7,false);
    }
    public Insets insets() {
        return new Insets(10,10,10,10);
    }
    public StringBuffer getEmployees(String item) {
        String department=item;
        Socket echoSocket = null;
        DataOutputStream outstream = null;
        DataInputStream instream = null;
        DataInputStream stdin = new DataInputStream(System.in);
        StringBuffer buf = new StringBuffer();
        StringBuffer bufback = new StringBuffer();

        try {
            echoSocket = new Socket("nthost.mydirectory.com", 80);
            outstream = new DataOutputStream(echoSocket.getOutputStream());
            instream = new DataInputStream(echoSocket.getInputStream());
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: nthost");
        } catch (IOException e) {
            System.err.println
                        ("Couldn't get I/O for the connection to: nthost");
        }
        if (echoSocket != null && outstream != null && instream != null) {
            try {
                String userInput;
                userInput = "GET /gotdata.asp?DSN=nthostempdata&DEPT="
                                                        + department;
                outstream.writeBytes(userInput);
                outstream.writeByte('\n');
                String fromServer;
                String goodLine;
                String[] firstname = new String[50];
                                        //50 occurrences of firstname
                String[] lastname = new String[50];
                                        //50 occurrences of lastname
                String[] salary = new String[50];//50 occurrences of salary
                String saveString;
                StringBuffer saveBuffer = new StringBuffer();
                int flagByte;
                int delimiter;
                int olddelimiter;
                char nameSeparator = '|';
                while ((fromServer = instream.readLine()) != null) {
                        flagByte = fromServer.indexOf('|');
                        if (flagByte > 0) {
                                buf.append(fromServer.substring
                                                (flagByte + 3) + "\n");
                        }
                }

                //parse occurrence 1 of the result set

                    goodLine = buf.toString();
                    int len = goodLine.length();
                    delimiter = goodLine.lastIndexOf('$');
                    salary[0] = goodLine.substring(delimiter + 1, len - 1);
                    buf.setCharAt(delimiter, ' ');
                                        //space over where comma was9
                    olddelimiter = delimiter;
                    goodLine = buf.toString();
                    delimiter = goodLine.lastIndexOf('$');
                    lastname[0] = goodLine.substring
                                        (delimiter + 1, olddelimiter);
                    buf.setCharAt(delimiter, ' ');
                    olddelimiter = delimiter;
                    goodLine = buf.toString();
                    delimiter = goodLine.lastIndexOf('$');
                    firstname[0] = goodLine.substring
                                        (delimiter + 1, olddelimiter);
                    buf.setCharAt(delimiter, ' ');
                    int checkdelimiter = delimiter;

                //parse occurence 2 et.al.

                    int subscript = 1;
                    while (checkdelimiter > 0) {
                       olddelimiter = delimiter;
                       goodLine = buf.toString();
                       delimiter = goodLine.lastIndexOf('$');
                      salary[subscript] = goodLine.substring
                                        (delimiter + 1, olddelimiter);
                       buf.setCharAt(delimiter, ' ');
                       olddelimiter = delimiter;
                       goodLine = buf.toString();
                       delimiter = goodLine.lastIndexOf('$');
                     lastname[subscript] = goodLine.substring
                                        (delimiter + 1, olddelimiter);
                       buf.setCharAt(delimiter, ' ');
                       olddelimiter = delimiter;
                       goodLine = buf.toString();
                       delimiter = goodLine.lastIndexOf('$');
                     firstname[subscript] = goodLine.substring
                                        (delimiter + 1, olddelimiter);
                       buf.setCharAt(delimiter, ' ');
                       checkdelimiter = goodLine.lastIndexOf('$');
                       subscript++;
                    }
                    for (int i = (subscript - 1); i > -1; i--) {
                       bufback.append(firstname[i] + "  " +
                                lastname[i] + "  " + salary[i] + " " + "\n");
                    }
                    outstream.close();
                    instream.close();
                    echoSocket.close();
                } catch (IOException e) {
                    System.err.println
                                ("I/O failed on the connection to: nthost15");
                }
            return bufback;
            }
        return bufback;
        }
        public Dimension minimumSize() {
            return new Dimension(150,130);
        }
        public Dimension preferredSize() {
            return minimumSize();
        }
}
```

LISTING THREE

```
<%

dbms = Request.QueryString("DSN")
dept = Request.QueryString("DEPT")

Set Conn = Server.CreateObject("ADODB.Connection")
Set RS = Server.CreateObject("ADODB.RecordSet")
Conn.Open "DSN=" + dbms + ";UID=;PWD="
temp="SELECT firstname, lastname, salary from EMPLOYEE WHERE " & _
    "(department IN ('" & dept & "'))"
set rstemp=Conn.execute(temp)
RS.Open temp, Conn
%>

<% = RS.Fields.Count %>|<% = RS.RecordCount %>
<%
Do While Not RS.EOF
    For cnt = 0 To (RS.Fields.Count - 1) Step 1
        OUT = "$ " & RS.Fields.Item(cnt)
        Response.Write(OUT)
        Next
        RS.MoveNext
Loop
  %>

  <%
RS.Close
%>
```

must be redisplayed to the user. That is the job of the `repaint()` method. Voilà! The user is now looking at the list of employees/salaries in department XYZ.

The `getEmployees()` Method

As you probably suspected, the real work takes place in the `getEmployees()` method. First, this method opens a connection to my host on the server by instantiating a `Socket` called `echoSocket`. The parameters are the name of the host machine (nthost.mydirectory.com) and the server port (80). Of course, you'll need to replace my host name with the name of your host machine. The port number usually stays the same for most configurations. After opening the connection, the next two lines instantiate objects for the output and input streams, respectively. These objects can be seen as two "pipes" that I have opened to and from the server. To get the server to do something, I will send it a request using `outstream`. Similarly, the server will send me back the results of the request via `instream`.

The next activity is to format the request to send to the server. The `userInput` statement in the `try` block invokes the name of an Active Server Page on the server. The first thing the command does is issue a `GET` request. This particluar `GET` says, "I am passing you the parameters as a series of variables that I will append to the ASP program I want to invoke." The next bit of information is the name of the ASP program I want to invoke; "getdata.asp" followed by a question mark. The two variables I want to pass to the ASP program are `DSN`, which is the name of the system-level Access database on the server; and `DEPT`, which is the department code the user selected. I move that entire string to the `userInput` variable.

Remember that output stream (pipe) we opened? I send the formatted request to `outstream.writeBytes(userInput)`. It simply means, "Write the command stored in the `userInput` variable to the `outstream` object." Off it goes to the server! The ASP program invoked by our request queries the Access database and returns the result set in the input-stream object called `instream`. All our Java program needs to do is read from that `instream` object to get the qualifying employees and their salaries.

Due to the way the ASP program was designed, the entire result set is returned as a single input stream. Each qualifying employee name (first and last) and associated salary information is followed by the next employee name and salary. So, in addition to reading the result set, I must parse it to display individual rows in the text area. The remainder of Listing Two mostly contains the parsing routine, which reads the entire stream starting from the end and moves back to the beginning looking for occurrences of the dollar character ("$"). I used the "$" character in the ASP program to delimit each occurrence of first name, last name, and salary.

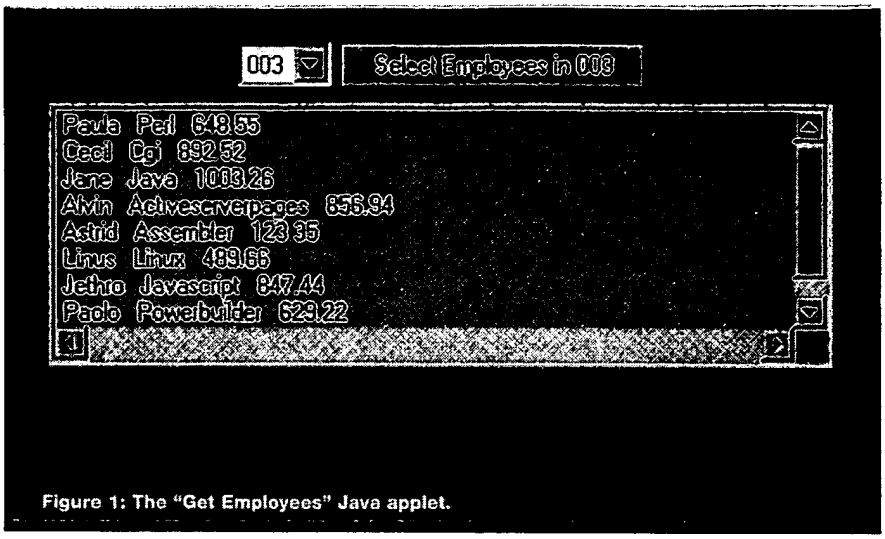


Figure 1: The "Get Employees" Java applet.

When reading the input stream, a `do-while` construct starts by reading a line from the `instream` object. The `readline()` method reads this line, which is then transferred to the `fromServer` character-string variable. Because the result set returned contains some header information I want to bypass, I read past it until I reach the pipe character ("|"), which signals the start of the employee data. This result set is moved to the `buf` variable for parsing. Basically, I index through the `buf` string looking for delimiters ($) between fields. Because I know the order of the fields (from the SQL issued in the ASP program), I can build arrays of first name, last name, and salary from the `buf` object. The final step is to format the object and return.

The Active Server Page

Listing Three contains the ASP application "getdata.asp." The ASP code does not have to be compiled prior to run time, but it does have to reside on the server. In our example, it is stored quite unceremoniously with everything else in the root directory: hence the simple call to /getdata.asp.

The first thing Listing Three does is retrieve the two parameters sent in the request: `DSN` and `DEPT`. The first two lines contain the commands to move the values from the two parameters to two corresponding variables in the program. `Request.QueryString` is an ASP command that simply reads the corresponding variable. Listing Three then opens a connection to the server, and instantiates a record set object `RS` that will be used to store the results of the SQL query. `Conn.Open` opens the database associated with the variable `DSN` (remember, this was passed from Listing Two as "NTHOSTempdata"). Next, I format the SQL that will be issued to the database. In our case, it is a simple `select` of first name, last name, and salary. Notice that the department code that was passed to the ASP program from Listing Two is embedded as `dept`. Listing Three then executes the SQL and returns the result set to the `rstemp` variable. `RS.Open` opens the result set and makes it available for processing. I then embed a "|" character preceded by the number of individual fields (or columns) retrieved from the database. A `do-while` loop simply loops through each field in the result set and writes it out to the output data stream. Finally, Listing Three closes the result set `RS`.

Conclusion

I have barely scratched the surface when it comes to the immense capabilities of Java and ASP scripting. However, this example illustrates the basics of what you can do with Access, Java, and ASP. Volumes can be written on each, of course. Hopefully, you can use the application template presented to take advantage of the considerable capabilities of each technology.

Art develops and maintains IBM/OS390 COBOL applications for a Fortune 500 company. He can be reached at acasale@packet.net.

U.S. DEPARTMENT OF COMMER
Patent and Trademark Off

DOCUMENT RETRIEVAL REQUEST FORM 149

Please include RightFax Number to expedite return of documents

Requester's Name: Lucas D. Vine	Case Serial Number: 09/709486		Art Unit/Org.: 2624
Phone: 27432	**RightFax:	Building: Knox	Room Number: 9D25
Class/Sub-Class:			
Date of Request: 7/21/05		Date Needed By: 7/22/05	

Paste or add text of citation or bibliography: **Paste Citation** Only one request per form. Original copy only. ☐

Author/Editor:	
Journal/Book Title:	
Article Title:	
Volume Number:	Report Number: Pages:
Issue Number:	Series Number: Year of Publication:
Publisher:	
Remarks:	

Staff Use Only

Monthly Accession Number:

Library Action	PTO		LC		NAL		NIH		NLM		NIST		Other	
	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd
Local Attempts														
Date	7/21													
Initials	KeT													
Results	Comp													
Examiner Called														
Page Count														
Money Spent														

	Source	Date
Ordered From:		

Remarks/Comments
1st and 2nd denotes time taken to a library

O/N – Under NLM means Overnight

Comments:

Subfile: C
Copyright 2000, IEE

...Abstract: The base framework uses today's most popular technologies, such as a three-layer framework, **Microsoft ASP** technology, and so on. This paper also gives examples from other colleges.
...Identifiers: **Microsoft ASP** technology

13/3,K/3 (Item 3 from file: 2)
DIALOG(R)File 2:INSPEC
(c) 2005 Institution of Electrical Engineers. All rts. reserv.

6479057
Title: Messaging in the next millennium
Author(s): Anderson, R.
Journal: Network Computing vol.10, no.25 p.80-2, 84, 86, 88
Publisher: CMP Media Inc,
Publication Date: 13 Dec. 1999 Country of Publication: USA
CODEN: NETCF7 ISSN: 1046-4468
SICI: 1046-4468(19991213)10:25L.80:MNM;1-C
Material Identity Number: H327-2000-001
Language: English
Subfile: D
Copyright 2000, IEE

Abstract: The big three integrated messaging and collaboration software makers, Novell, Lotus and **Microsoft** , will revamp their applications. New versions of GroupWise, Notes and Exchange will focus on reliability and scalability to set the stage for a push into the ISP/ **ASP** arena. Look for software that's more customizable than ever, with greater Web and mobile...
...Identifiers: **Microsoft** Exchange

13/3,K/4 (Item 4 from file: 2)
DIALOG(R)File 2:INSPEC
(c) 2005 Institution of Electrical Engineers. All rts. reserv.

6455978 INSPEC Abstract Number: C2000-02-7210L-038
Title: Web-based access to locally developed databases
Author(s): Mischo, W.H.; Schlembach, M.C.
Author Affiliation: Grainger Eng. Libr. Inf. Center, Illinois Univ., Urbana, IL, USA
Journal: Library Computing vol.18, no.1 p.51-8
Publisher: Sage Publications,
Publication Date: 1999 Country of Publication: USA
CODEN: LICOFW ISSN: 0742-5759
SICI: 0742-5759(1999)18:1L.51:BALD;1-A
Material Identity Number: H418-1999-001
U.S. Copyright Clearance Center Code: 0742-5759/99/$0.50+.10
Language: English
Subfile: C
Copyright 2000, IEE

...Abstract: Champaign has implemented Web-based access to locally developed information database resources. The systems use **Microsoft Active Server Pages** (**ASP**) technology. **ASP** is a convenient and powerful alternative to CGI (Common Gateway Interface) scripting, for Web database...
... also have improved library staff training and knowledge of the

Web-Based Access to Locally Developed Databases

William H. Mischo and Mary C. Schlembach

The Grainger Engineering Library Information Center at the University of Illinois at Urbana-Champaign has implemented Web-based access to locally developed information database resources. The systems use Microsoft Active Server Pages (ASP) technology. ASP is a convenient and powerful alternative to CGI (Common Gateway Interface) scripting, for Web database connectivity. The local databases complement traditional library resources such as the on-line catalog and commercial databases. They also have improved library staff training and knowledge of the collections. This article describes the ASP application, the database structure approach, and the Web technologies for providing Web-based searching of multiple databases.

The World Wide Web is the new "gold standard" interface for providing access to on-line information resources such as locally accessible on-line public access catalogs (OPACs), article index databases, and full-text document (publisher) products.

"Without a doubt the World Wide Web is now the vehicle of choice for online databases and other collections of online information" (Notess 1998). Web-based resources offer many advantages over telnet and CD-ROM access. The Web offers a consistent search, retrieval, and display paradigm independent of the hardware or operating system being used. Additionally, libraries have begun to offer or are experimenting with Web-based access to locally produced information resources formerly available in vertical files, word processing documents, or within client-server database systems accessible through custom client software (Perez 1998a; Jacsó 1998).

A number of libraries now provide access to local information files within independent telnet environments or as part of the local OPAC configuration (Horah 1998; Perez 1998b). Some organizations have converted their paper vertical files to a Web format by creating Web pages that contain hypertext links to the Web sites of the organizations or entities formerly represented in the vertical file (Neuhaus 1997; Colburn 1997).

The University of Illinois at Urbana-Champaign (UIUC) Grainger Engineering Library Information Center provides access to a number of custom-developed locally created databases within an integrated Web-based environment. These Web-based databases are designed to provide patrons and reference staff with enhanced access to important information resources and also to assist in the training of reference staff, particularly graduate assistants and paraprofessional staff. This article will describe the Web-based technologies employed by Grainger staff in implementing access to local information resources. It includes a discussion of Microsoft (MS) Active Server Pages (ASP) technologies and the associated local database structure and format. Also discussed is the general adaptability of these techniques.

Local Databases

Several of the local databases available on the Grainger Web sites were originally developed as word processor or text files for printed lists. These files were converted into MS Access databases using custom Visual Basic programs written by Grainger staff. Other databases have been generated using custom Visual Basic data entry and verification programs or program-based data extraction from other resources. Many of the MS Access databases have been

Library Computing, Vol. 18 No. 1, 1999 51-58

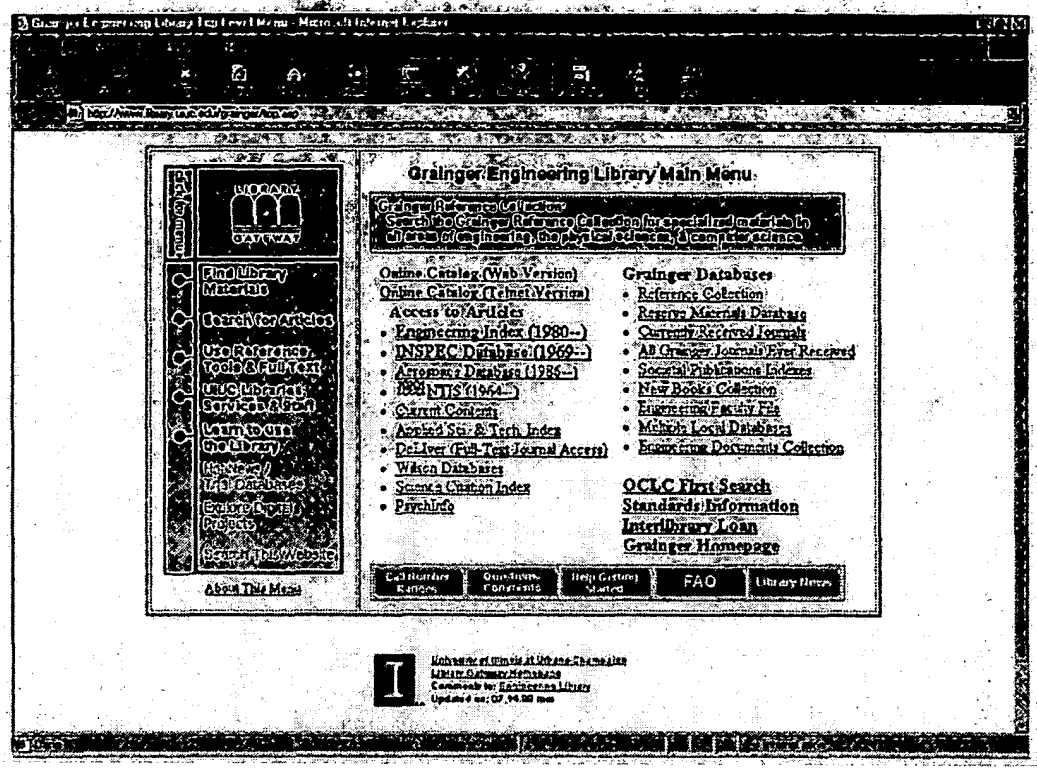


Figure 1. Grainger public terminal top-level menu.

moved to MS SQL Server to ensure better transaction throughput, particularly in multiuser situations.

The UIUC libraries have a long commitment to providing access to locally generated microcomputer databases at public workstations. Beginning in 1990, PC public terminals with custom interface software provided access to local resources—at that time either stored on each machine or, later, available over the building network (Mischo and Cole 1992). As we moved to a client-server environment with networked databases, each machine required a custom Visual Basic client to retrieve information from the databases. All of the Grainger local databases have now been moved to a Web environment using MS ASP technologies for Web-based retrieval. Local database access in Grainger has thus followed a natural evolution from stand-alone microcomputer-based systems to local-area networked architectures to a Web-based environment.

These local resources are available from the Grainger Library home page, the Web top-level menu on Grainger public terminals, and a custom reference desk Web page. Figure 1 shows the public terminal top-level menu. Placing the mouse over any link brings up a short description of the resource. Figure 2 shows the database selection page that is one link below the Grainger home page.

Among the custom Web-based databases available through the Grainger Library Web are

- a Reference Collection database with added subject access points taken from each work's table of contents as well as annotations and comments entered by Grainger staff to indicate the most appropriate reference tool to meet a patron-specific information need;
- a Difficult Citations and Frequently Asked Questions database of both commonly asked questions and difficult-to-find information and holdings/locations;
- a multisociety professional engineering societies publications database providing simultaneous searching of society publications;
- a currently received journals database including check-in and binding information and links to tables of contents of specific issues via a custom component that connects to the Current Contents database running under Ovid;
- a database of all journals ever received in the Engineering Library with previous titles and succeeding titles;
- a Reserve Collection database, which includes links to instructor-supporting materials on the Web;
- a UIUC College of Engineering Documents Center collection of uncataloged technical reports authored by UIUC faculty;
- a New Books database with links to online catalog records and order records;
- a Faculty Research Interests database with information on individual faculty research accomplishments, awards, honors, and so on; and

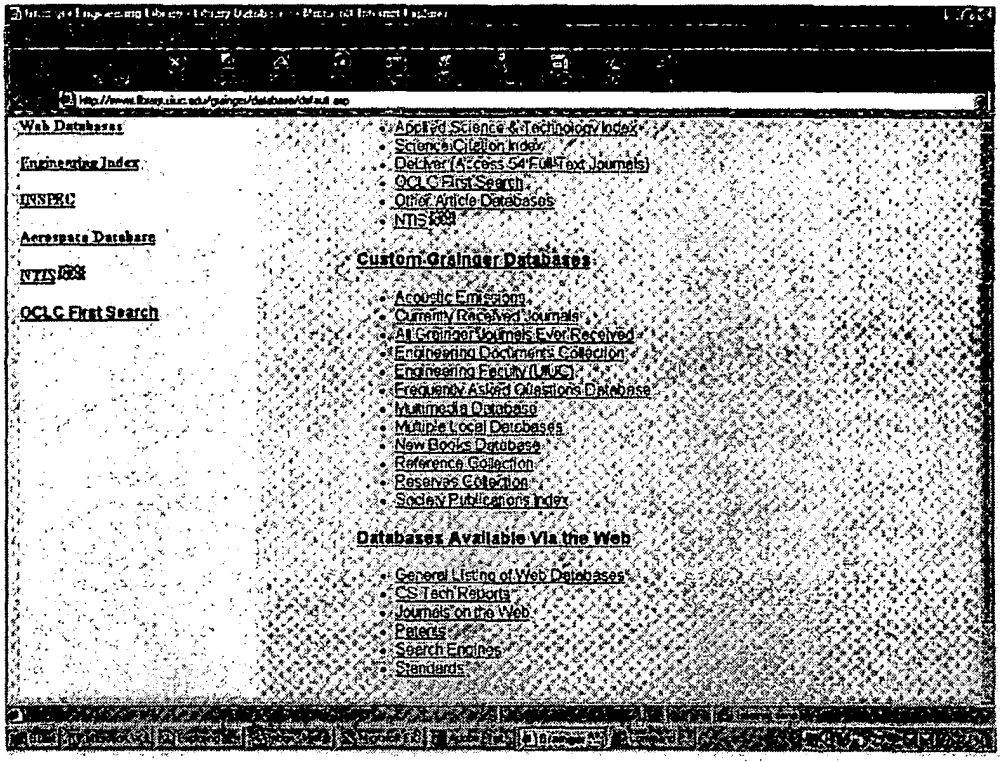


Figure 2. Grainger home page custom databases.

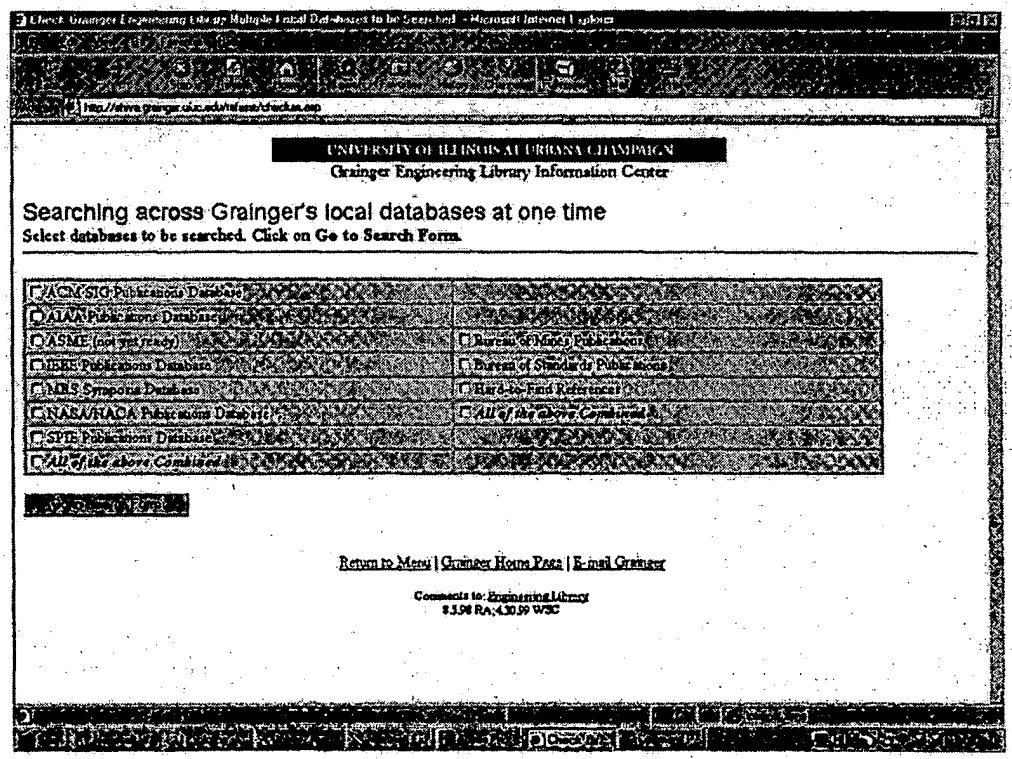


Figure 3. Multiple local database selection menu.

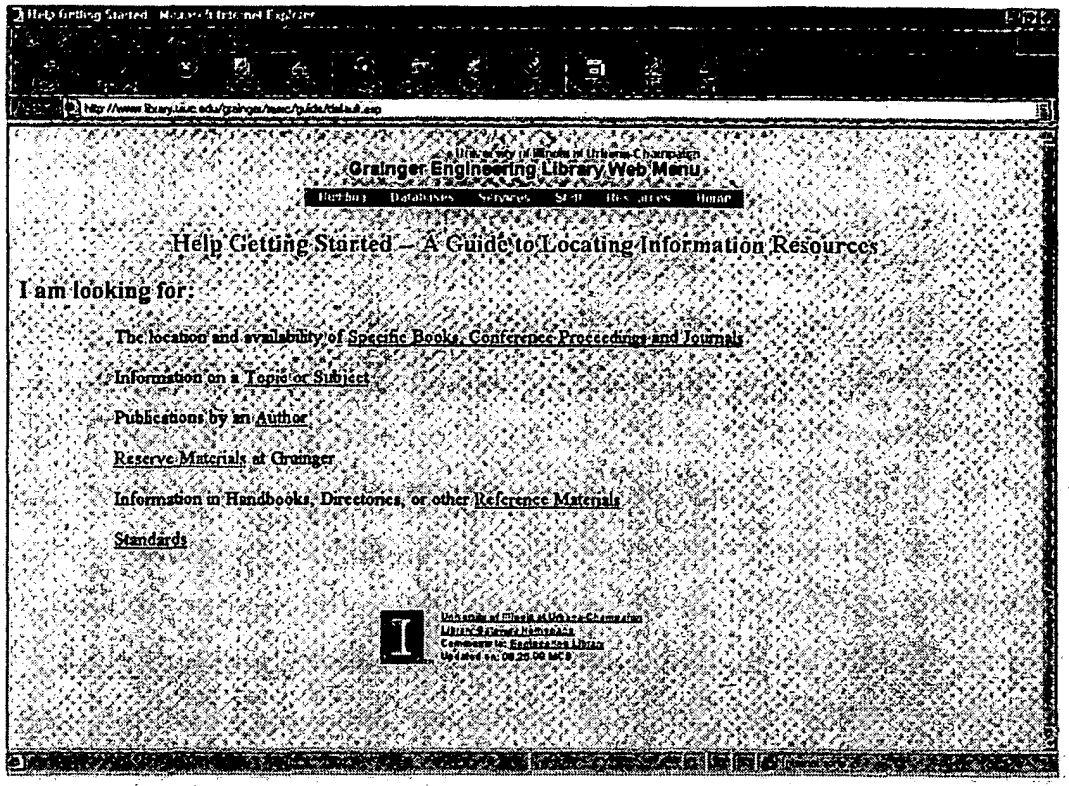


Figure 4. Help Getting Started top-level menu.

- a Multiple Resources database that allows simultaneous, discrete searching of a number of the local files (see Figure 3).

Altogether there are currently twenty locally produced Web-accessible databases, containing anywhere from several hundred records to tens of thousands of records.

The Web-based local resources complement or supplement the on-line catalog and other available information resources. They provide expanded access to materials (such as in the Reference Collection database), they increase access to difficult-to-find resources (such as serial holdings), and they provide access to materials without more traditional bibliographic access points, such as uncataloged engineering technical reports.

As the number of available information resources continues to grow, Grainger staff have developed a "Help Getting Started" module, the first page of which is shown in Figure 3. This helps users to identify the most appropriate resources to meet their information needs.

Web Technologies

The Grainger local databases that have been made available over the Web have been developed under the Active Server Platform or Active Server Pages (ASP) technology on MS Internet Information Server (IIS) running under Windows NT. ASP provides developers with server-side scripting tools to create dynamic Web pages. ASP uses predefined component objects to connect to ODBC databases, such as MS Access or MS SQL Server and was designed as a "convenient alternative to conventional CGI scripting with Perl or C" (Garris 1998).

These Web pages, which have an .asp file extension, provide browser-neutral database connectivity over the Web. ASP files contain a mix of HTML and scripting code written in VBScript, JavaScript, or PerlScript. Within an ASP file, the scripting code is executed on the server with the resulting code converted to HTML, which is sent directly to the browser. ASP adds options, such as the maintenance of stateful connections to ODBC databases, the generation of dynamic content, the ability to read and write text files, the automatic conversion of scripting code to HTML, the capability of using internal and external program components, and the ability to define global variables within a session.

ASP executes within the current computing process and does not incur the processing overhead of CGI (Common Gateway Interface). CGI calls start a separate process for every new remote request to the server. Web servers that use in-process Application Programming Interfaces (APIs),

such as ASP, offer significant advantages over CGI. The performance advantage is particularly noticeable at high transaction levels, with large numbers of simultaneous users. Recent tests conducted by *PC Magazine* show that in-process APIs perform better than CGI, by a factor of 4 to 1 (Seltzer 1999).

ASP has thus proven to be an extremely productive and popular tool: "if you're a (Web) developer searching for a Holy Grail, then IIS and Asp could end your quest" (Alwang 1997).

High-volume e-commerce sites are heavy users of ASP. For example, Dell Computer, Microsoft, Barnes and Noble, and MSNBC are all ASP users. Although ASP is most commonly used in a Microsoft IIS environment, ASP is now commercially offered by third-party vendor Chili!Soft to operate on other NT Web servers, such as Netscape's, and for several UNIX platforms, including Sun Solaris and IBM AIX (Carey 1999).

IIS and ASP thus provided us with a convenient migration path for moving the local databases and clients that we had constructed over to a Web-based retrieval environment. Many of the retrieval and interface techniques originally developed for Visual Basic client-server applications can be transferred to the ASP environment. Both Visual Basic and VBScript within ASP require knowledge of an object-oriented scripting language and the ability to formulate effective SQL commands. In particular, we were able to effectively transport the routines for input string parsing and the appropriate SQL commands from Visual Basic over to ASP.

There was one minor obstacle. In 1997, there were no useful rapid development tools for ASP; therefore, we were forced to become "code warriors," using simple editing tools like Notepad. Currently, however, there are numerous Web rapid application development (RAD) tools, such as Visual InterDev, FrontPage, and others, which offer advanced features like scripting and SQL construction support.

Our Web applications are typically composed of several ASP pages, with search argument entry forms separated from the results display pages. Although it would be fairly easy to combine all tasks into a single ASP page, we have opted to treat each application task as a separate page module. This is handy for debugging purposes and logical structure. Our display pages are built dynamically and recursively so that a single page, which posts to itself, is used to build the displayed records.

Web development tools providing database connectivity are server based. They typically use either CGI scripting or a page-based technology such as ASP for their processing. CGI requires knowledge of Perl or C. The ASP scripting language is quite a bit easier to learn and offers built-in tools for common functions, including database connectivity. As mentioned earlier, there are significant performance benefits with ASP, particularly for small to midsize applications.

Coming from a Visual Basic background, we had easy transition to ASP. We also have had success in teaching ASP techniques to Library School students, in a one-semester class on distributed information systems. This class requires students to complete a final database project utilizing MS Access and ASP. The steepest learning curve for the students is in mastering the SQL query language, not in picking up ASP. Overall, we have found that combining ASP techniques with a standard relational database structure has resulted in a robust development environment.

Database Structures and Retrieval Features

Our custom databases employ a standard database structure and data element format. These conventions have been applied across most of our local database resources. This standard database structure includes a common table and field (column) tags and rules for formatting field data. For example, the standard "Authors" tag is uniformly used in each database. Multiple author names are entered consecutively within a single field in the form: last name, comma, first name/initial, with terminating semicolon delimiter, followed by a space. This allows for accurate author searching with no cross-name hits or reversed first name–last name retrievals.

The defined fields within each file are a portion of the standardized field superset covering the entire collection. Each database file uses appropriate fields, as applicable. This provides advantages in search and retrieval operations. It allows simultaneous searching of multiple databases with or without the construction of merged databases. The common table and field names and consistent format of the data further allow the creation of general purpose ASP routines for building the appropriate SQL commands needed to search selected database(s).

Graduate assistants are primarily responsible for maintenance and updating of the local databases. We use two methods to do this: data entry forms using Web-based or Visual Basic interfaces, and program-based data extraction from other resources such as the on-line catalog or article databases.

On the Web side, the Grainger local databases employ the same common interface look and feel for search and display operations. The initial .asp Web page for each resource displays a "query-by-example" entry form, using query fields appropriate to the particular database file.

The databases consistently feature

- the ability to search multiple fields and utilize Boolean OR or AND operators;
- a choice of displaying search matches one full record at a time or fifteen abbreviated records at a time with links to the full record;

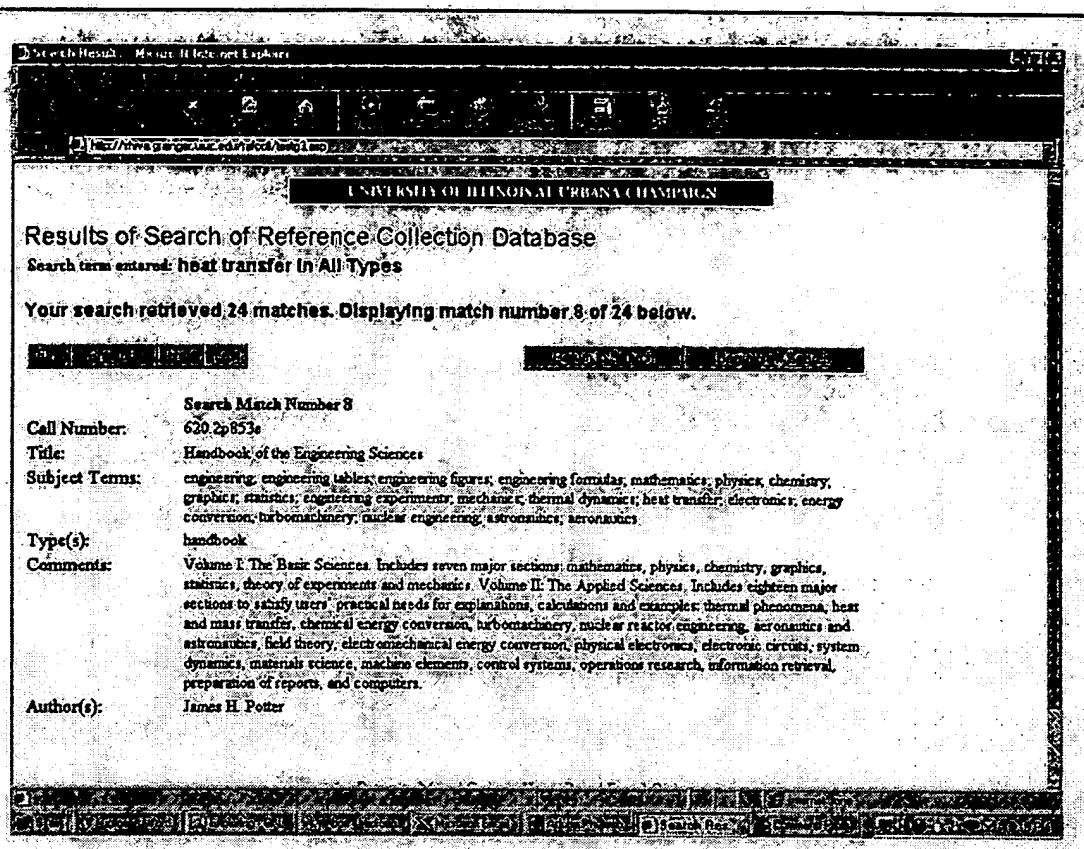


Figure 5. Record navigation buttons.

- default keyword "AND" searching for multiword search arguments;
- default left- and right-hand truncation of search arguments;
- exact phrase searching using quotation marks;
- separate prompts for author last name and first name;
- Next, Previous, First, and Last record navigation buttons (Figure 5); and
- return to the original top-level menu.

See Figures 6 and 7 for the search entry forms from the Reference Collection database and the Engineering Documents Center first page. Special search features, such as limiting to a specific type of reference tool, are also offered for specific databases, as appropriate.

Programming Sophistication

Our ASP pages all employ reusable modules within the different databases. They share the same ASP code base, with modifications made for the specific database. We are now developing a general purpose template program in ASP. The template program will prompt a database designer for parameters such as database name, field names, search field labels, and so on, and then automatically generate the ASP pages for that specific database application. The commercial software package Drumbeat 2000 (`www.drumbeat.com`) provides similar database design functionality. Such a tool will significantly speed up development of new database applications at the Grainger Engineering Library.

We also expect to expand our simultaneous-search capabilities. We have already implemented this across the multiple ASP databases. We have now developed a Z39.50 dynamic link library (.dll) to allow cross-database searching of the library online catalog, selected periodical index databases, and our local databases. A single search query form built using ASP technology will handle this global search function.

We also have been exploring multimedia search techniques, including voice input and output and video, in providing intelligent and flexible point-of-contact search navigation for end users and reference staff. This is part of a project called Reference Assistant. We are designing Reference Assistant to give end users access to the human knowledge base of the Grainger reference librarians.

Usage

Both library users and library staff are making heavy use of the local databases. This is substantiated by our public terminal log files. These transaction logs record search

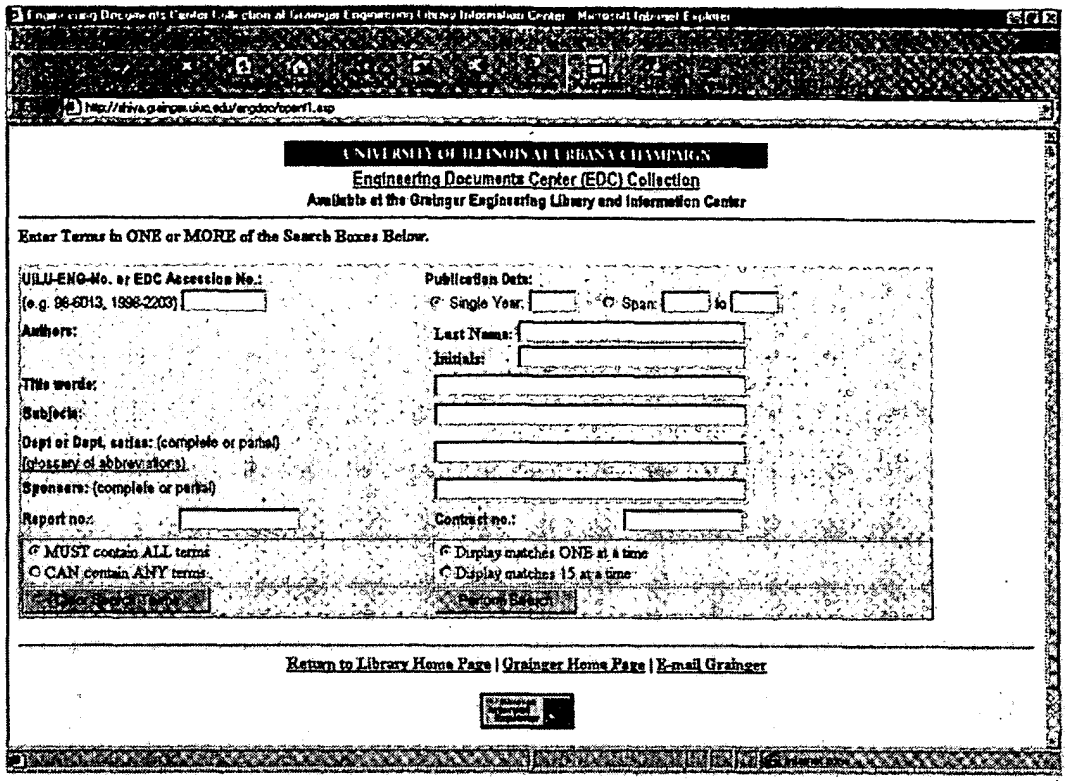


Figure 6. Engineering Documents search form.

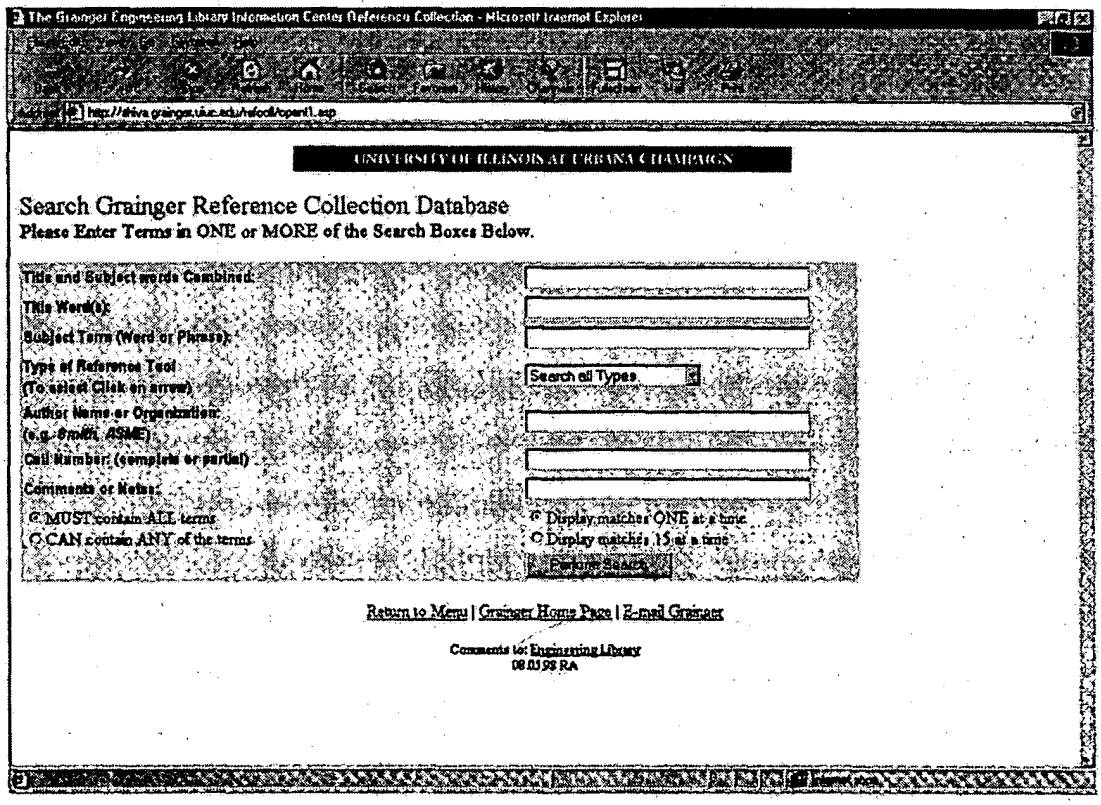


Figure 7. Reference Collection database search form.

activity from twenty public terminals. These logs are recorded using an .asp page that calls an external component that, first, writes the transaction to a text file and then redirects to the appropriate destination Web page. Our log data go back to December 1997 and cover user top-level menu selections (see Figure 1), including accesses to the online catalog, article index databases, OCLC FirstSearch, interlibrary loan, and various local resources. The logs show that local databases comprise approximately 24 percent of the public terminal main menu selections. In addition, the local databases are accessed an average of several hundred times per month from reference desk terminals and remote access to the Grainger Web pages.

Conclusion

The Grainger Library Web-based local databases provide greatly improved services to users both inside and outside the library. They also provide remote Web access service to our users when the Library is closed. Patrons use our databases to locate unique and hard-to-find materials or to locate information not available in our standard bibliographic resources.

In addition, the training of reference staff has been improved, with the availability of special database tools such as the Reference Collection database with its comments and notes, the Difficult Citations database, the Frequently Asked Questions database, and by our newfound ability to simultaneously search multiple local database resources.

References

Alwang, Greg. 1997. Review of microsoft internet information server. *PC Magazine* 16 (10): 187.

Carey, Theresa W. 1999. New stuff: Chili!Soft supports more platforms. *Microsoft Internet Developer* 4 (7): 10.

Colburn, Joan L. 1997. Horizontal wires replace the vertical files. *Medical Reference Services Quarterly* 16 (3): 19-25.

Garris, John. 1998. Active server pages: Scripting with ASP. *PC Magazine* 17 (12): 245-46.

Horah, Jan L. 1998. The evolution of a library database: From cards to the Web. *Searcher* 6 (5): 56-60.

Jacsó, Péter. 1998. Publishing textual databases on the Web. *Information Today* 15 (9): 47-48.

Mischo, William H., and Timothy W. Cole. 1992. The Illinois extended OPAC: Library information workstation design and development. In *Advances in online public access catalogs*, ed. Marsha Ra, 38-57. Westport, CT: Meckler.

Neuhaus, Chris. 1997. Developing a hypertext World Wide Web vertical file. *Collection Building* 16 (1): 66-72.

Notess, Greg R. 1998. The year databases moved to the Web. *Database* 21 (6): 56-58.

Perez, Ernest. 1998a. Supercharge your Web site. *Library Software Review* 17 (1): 24-30.

———. 1998b. Converting a card index backfile. *Database* 21 (6): 63-65.

Seltzer, Larry. 1999. Create a great site: Serve it. *PC Magazine* 18 (11): 172.

About the Authors

William H. Mischo is the engineering librarian and professor of library administration at the Grainger Engineering Library Information Center at the University of Illinois at Urbana-Champaign (UIUC). He earned his M.A. in library and information science from the University of Wisconsin–Madison. He can be reached at 217-333-7497; e-mail: `w-mischo@uiuc.edu`.

Mary C. Schlembach is an assistant engineering librarian for Digital Services at the Grainger Engineering Library Information Center. She earned her M.L.S. at the University of Illinois at Urbana-Champaign and a C.A.S in library automation from the University of Pittsburgh. She can be reached at 217-333-3158; e-mail: `schlemba@uiuc.edu`.

U.S. DEPARTMENT OF COMMER
Patent and Trademark Off

DOCUMENT RETRIEVAL REQUEST FORM

Please include RightFax Number to expedite return of documents

150

Requester's Name: Lucas Dixie	Case Serial Number: 09/709486		Art Unit/Org.: 2624
Phone: 27432	**RightFax:	Building: Knox	Room Number: 9D28
Class/Sub-Class:			
Date of Request: 7/21/05		Date Needed By: 7/22/05	

Paste or add text of citation or bibliography: Paste Citation Only one request per form. Original copy only. ☐

Author/Editor:	
Journal/Book Title:	
Article Title:	
Volume Number:	Report Number: Pages:
Issue Number:	Series Number: Year of Publication:
Publisher:	
Remarks:	

Staff Use Only

Monthly Accession Number:

Library Action	PTO		LC		NAL		NIH		NLM		NIST		Other	
	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd
Local Attempts														
Date	7/21													
Initials	REV													
Results	Cal													
Examiner Called														
Page Count														
Money Spent														

	Source	Date
Ordered From:		

Remarks/Comments
1st and 2nd denotes time taken to a library

O/N – Under NLM means Overnight

Comments:

5/3,K/1 (Item 1 from file: 16)
DIALOG(R)File 16:Gale Group PROMT(R)

06466397 Supplier Number: 54890933 (USE FORMAT 7 FOR FULLTEXT)
Developing with Internet Explorer 5.0.(Product Information)
Lam, John
PC Magazine, p258
July 1, 1999
Language: English Record Type: Fulltext Abstract
Document Type: Magazine/Journal; General Trade
Word Count: 2663

ABSTRACT:
...the Web server. Typically, this has been solved through Web server services, such as the **session objects** in **Microsoft** 's **Active Server Pages** . IE5, through its support for the Extensible Markup Language (XML), makes it possible for Web...
... and its associated data has typically been solved through Web server services, such as the **session objects** in **Microsoft** 's **Active Server Pages** . But this is hardly the only solution to this problem; in fact it is not...

5/3,K/2 (Item 1 from file: 47)
DIALOG(R)File 47:Gale Group Magazine DB(TM)

05387730 SUPPLIER NUMBER: 54890933 (USE FORMAT 7 OR 9 FOR FULL TEXT)
Developing with Internet Explorer 5.0.(Product Information)
Lam, John
PC Magazine, 258
July 1, 1999
ISSN: 0888-8507 LANGUAGE: English RECORD TYPE: Fulltext; Abstract
WORD COUNT: 2802 LINE COUNT: 00222

...ABSTRACT: the Web server. Typically, this has been solved through Web server services, such as the **session objects** in **Microsoft** 's **Active Server Pages** . IE5, through its support for the Extensible Markup Language (XML), makes it possible for Web...
... and its associated data has typically been solved through Web server services, such as the **session objects** in **Microsoft** 's **Active Server Pages** . But this is hardly the only solution to this problem; in fact it is not...

5/3,K/3 (Item 1 from file: 88)
DIALOG(R)File 88:Gale Group Business A.R.T.S.

05127484 SUPPLIER NUMBER: 54890933
Developing with Internet Explorer 5.0.(Product Information)
Lam, John
PC Magazine, 258
July 1, 1999
ISSN: 0888-8507 LANGUAGE: English RECORD TYPE: Fulltext; Abstract
WORD COUNT: 2802 LINE COUNT: 00222

...ABSTRACT: the Web server. Typically, this has been solved through Web server services, such as the **session objects** in **Microsoft** 's **Active**

Developing with Internet Explorer 5.0

By John Lam

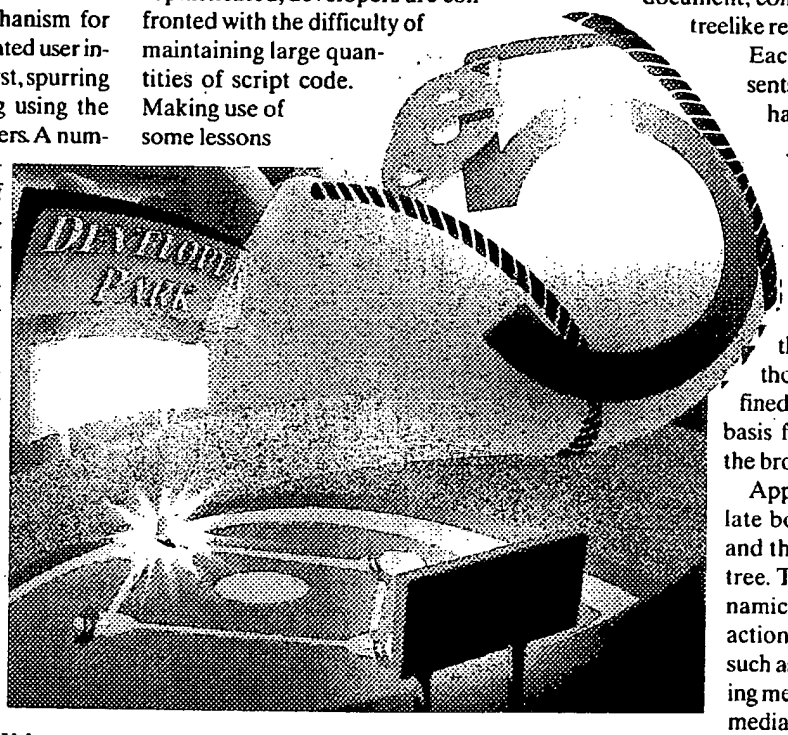
We know Web browsers are useful end-user applications, good for searching, browsing, and retrieving information. But until fourth-generation browsers introduced Dynamic HTML (DHTML), browsers provided few services for developers interested in creating Web-based applications. By offering a mechanism for building Web pages with sophisticated user interfaces, DHTML acted as a catalyst, spurring developers to begin considering using the Web to deliver functionality to users. A number of difficulties remained, however. We'll take a look at some of these problems and at the solutions Microsoft Internet Explorer 5 (IE5) provides, focusing on DHTML behaviors and XML support in IE5.

The first and most significant problem developers faced was maintaining a *user session* between the browser and the Web server. A user session might consist of, say, a shopping trip to Amazon.com, in which a user browses for books at the Web site and adds some to a shopping cart. In this case, the session data is the list of books in the shopping cart.

The problem of maintaining a user session and its associated data has typically been solved through Web server services, such as the session objects in Microsoft's Active Server Pages. But this is hardly the only solution to this problem; in fact it is not the best solution. IE5, through its support for the Extensible Markup Language (XML), makes it possible for Web developers to maintain a user session *and* its associated data all within a Dynamic HTML page on the client.

XML also serves as a means for dealing with a related issue: transferring data between the Web browser and the Web server. Until now, you had to transmit data using crude techniques such as HTML FORM elements. Moreover, if the data to be transferred was structured, developers had to invent ad hoc formats for representing the information. With XML, it's relatively easy to generate XML documents from an application's data; that data will then be processed by IE5's XML processor, saving considerable development effort.

Finally, as Web pages become increasingly sophisticated, developers are confronted with the difficulty of maintaining large quantities of script code. Making use of some lessons learned in building traditional Windows applications, IE5 introduces a feature called *DHTML behaviors*, which offers the benefits of component-based software development that Visual Basic and Delphi developers have grown accustomed to.



You can think of the new development support in IE5 as part of the ongoing evolution of Dynamic HTML. To understand how all the pieces fit together, let's first look at the role of Dynamic HTML in Web applications.

MAKING HTML DYNAMIC

Dynamic HTML lets you define a user interface in terms of HTML elements and lets users interact with that interface. DHTML consists of a number of related technologies: HTML defines the structure of the information being conveyed to the user; Cascading Style Sheets (CSS) determines the appearance of that information as it is being rendered; and the Document Object Model (DOM) lets scripting languages such as VBScript and JavaScript interact with the HTML elements in the document. Together, these components make it possible to write client applications that live inside the Web browser.

When IE5 requests an HTML document from a Web server, it establishes an HTTP connection to that server and retrieves the document. The browser then parses the document and retrieves any related information referred to by the document, such as embedded GIF images. In the process of parsing the document, IE5 creates an in-memory image of the document, converting it into an internal treelike representation (see Figure 1). Each node of the tree represents an HTML element that behaves as an independent object. These objects have properties and methods as defined by the Document Object Model Level 1 specification (available at *www.w3.org/tr/rec-dom-level-1*), and they can fire events, which—though this is not formally defined in the specification—is the basis for adding interactivity to the browser.

Application code can manipulate both the individual objects and the overall structure of the tree. This is what puts the "dynamic" in DHTML: Any interactions with individual objects, such as setting properties or calling methods, will be reflected immediately in the rendered document. Similarly, user interactions with individual objects, such as rolling the mouse over an object, will result in that object firing an event, which can be handled by application code.

When application code manipulates the tree itself, perhaps by adding or replacing nodes, the browser ensures that the document is correctly rerendered to reflect those changes. User interfaces can be built on the fly simply by building strings that contain HTML element sequences and having the browser parse the strings to modify the HTML object tree. Figure 2 illustrates this procedure, using JavaScript to add some HTML button elements to an existing <DIV>

element in the parsed HTML tree.

The Document Object Model forms the core of DHTML, defining properties and methods for individual objects within the HTML tree, as well as how individual object types (document, paragraph, or list element, for example) can be manipulated through script. It does so by defining one or more *interfaces* to each object type. An interface is simply a grouping of related methods and properties (such as the `IHTMLElement` interface that specifies properties and methods common to all HTML elements). In no way, however, does the DOM attempt to stipulate how the objects are to be implemented.

Microsoft chose to use the Component Object Model (COM) to implement the interfaces to the HTML tree objects. This is a good fit, since COM lets developers separate an object's *interface* from its *implementation*: The World Wide Web Consortium defines the interfaces, and Microsoft implements those interfaces for the HTML elements that can be manipulated by the browser.

An important consequence of using COM to implement the interfaces is that Windows developers can leverage their existing knowledge of COM programming. All they have to understand is how to call methods on COM objects from their favorite programming language or environment. All modern development tools for the Windows platform support calling methods on COM objects, as do Visual Basic, Delphi, Visual C++, and even the Microsoft Java Virtual Machine. And the scripting languages used in Dynamic HTML—VBScript and JavaScript—can also call methods on COM objects. In fact, when you create DHTML pages and manipulate HTML elements on those pages, you're really just calling methods on the underlying COM objects that make up those pages in memory.

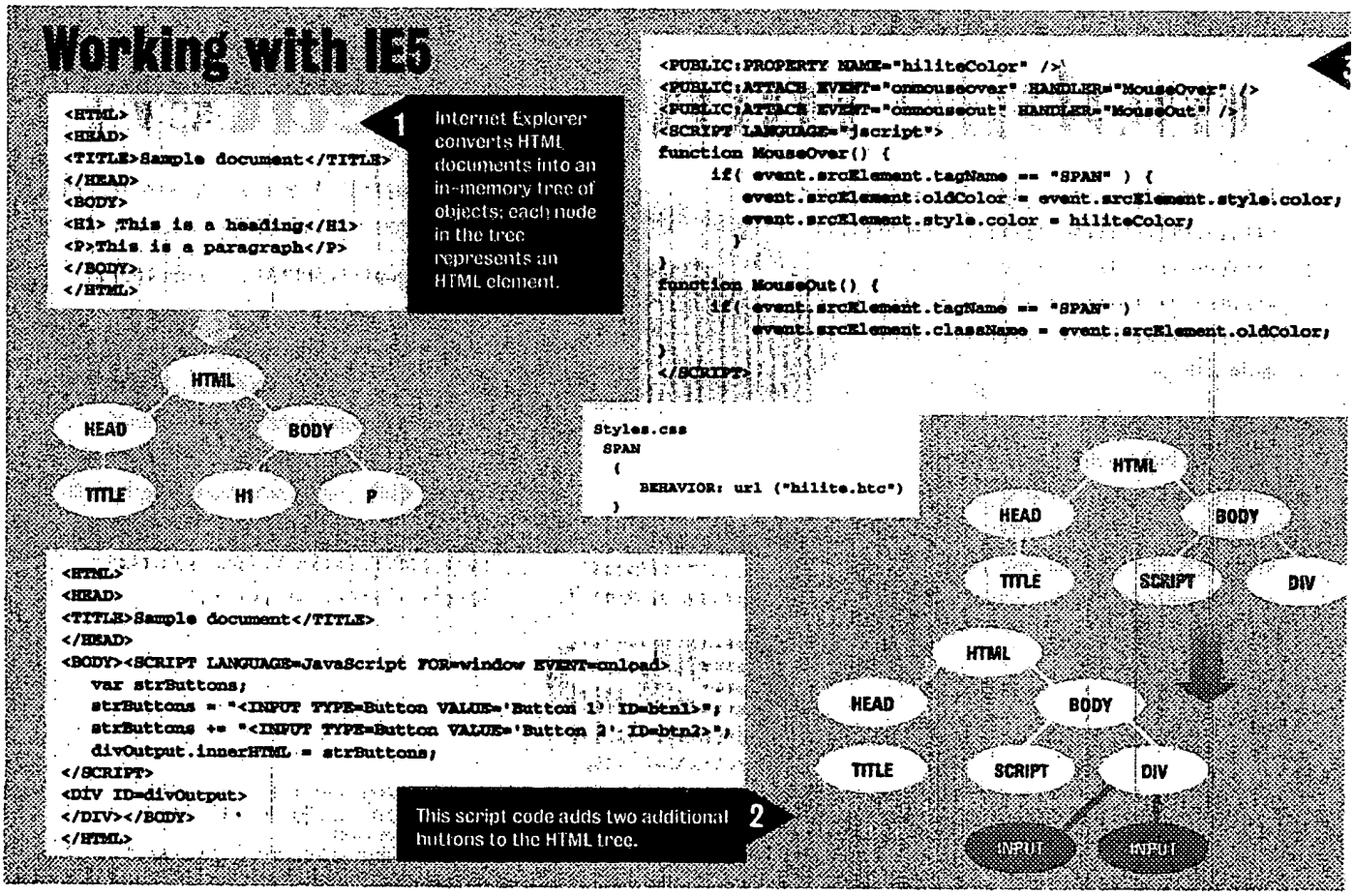
ELEMENTS OF STYLE

What defines the appearance of the individual nodes of the HTML tree? The answer is Cascading Style Sheets (CSS). Before CSS, you had to manipulate the property settings of objects individually. That meant a lot of redundant work and opportunity for error. With CSS, on the other hand, you define a sequence of formatting *rules* that the browser applies to elements of the HTML tree each time it renders the page.

CSS rules are applied in a specific order, with more specific rules overriding less specific rules. HTML elements that are children of other HTML elements will inherit the rules of their parents, unless there are rules in the CSS stylesheet that cause the parent's rules to be overridden. You tie CSS rules to an HTML document by inserting a link in the document to an external file that contains those rules. You can thus define the appearance of not just one page, but entire collections of pages simply by having all of those pages refer to the same CSS file. Not only does this make content more manageable, it also means the client can cache the referenced stylesheet, significantly reducing the amount of bandwidth required to render a page.

CODES OF BEHAVIOR

What began as a solution for managing the appearance of elements turned out to be much more. With the introduction of DHTML behaviors in IE5, Cascading Style Sheets can also be used to define rules that associate code



with specific HTML elements on a page.

Just as CSS lets you separate an HTML element's content from its appearance, DHTML behaviors separate the element's content from the code that specifies the element's behavior, functioning much as ActiveX controls do in traditional programming environments. Not surprisingly, DHTML behaviors are applied to the document using CSS rules. Let's look at an example.

You can write a DHTML behavior entirely in script. Such behaviors are known as *hypertext components* in IE5 and are stored in external script files with an .htc extension. They are "hooked up" at runtime to specific HTML elements, as laid out by the rules in a CSS stylesheet attached to an HTML document. Figure 3 shows a simple hypertext component file that attaches itself to an HTML element and highlights the text within that element whenever the mouse moves over it. This behavior receives both the onmouseover and onmouseout events from the SPAN element to which it is bound. It also exposes a property that lets the programmer specify what color the highlight should be when the mouse moves over the element.

Figure 3 also shows an excerpt from the associated CSS file, illustrating how IE5 extends CSS stylesheets with the new behavior attribute in which you specify the URL containing the hypertext component file. As you can see, just you can define the appearance of HTML elements, now you can define the behaviors of HTML elements, and not only for a single document, but across an entire family of related documents.

DHTML behaviors aren't limited to script-based hypertext components; they can be written in any language that supports COM. COM objects are typically referenced from HTML documents with the <OBJECT> tag. Note that a COM object, because it is compiled code, must first pass the security challenges IE5 presents (depending on the user's security settings) before it is allowed to download or execute within the user's browser.

There is a real advantage to creating DHTML behaviors in compiled code rather than script: You can protect your code investment, because you don't ship source code with your applications. Furthermore, you can potentially gain significant performance improvements using a compiled language. Finally, compiled languages give you full access to the underlying operating system in cases where you need tighter integration with your user's computer.

Though DHTML behaviors are a new feature, many samples are already available on Microsoft's Web site, at *msdn.microsoft.com*.

DEALING WITH DATA

Most substantive applications will deal with data and, as we noted, the transferring of data between browser and server has been one of the most common problems facing developers. Over the past year, Extensible Markup Language (XML) has emerged as a solution. IE5 adds an excellent XML parser, which developers can manipulate, to the object model within the browser. By interacting with the XML parser from their DHTML pages, applications can transfer data in the form of XML documents in both directions.

An XML document looks much like its HTML counterpart; each contains a set of tags that describe the information within the file. As with HTML documents, the tags define a tree of objects that is generated when an XML parser reads an XML document. This in-memory tree can be modified at runtime.

Figure 4 shows an XML document containing a list of objects, each representing an all-

This hypertext component highlights the text of any SPAN element it is bound to. The color is specified by the hiliteColor property and the binding is controlled by the styles.css stylesheet below).

```
<XML ID=xmlData SRC="getallstardata.asp?league=American"></XML>
<DIV ID=divBtn LANGUAGE=JavaScript onclick="return divBtn_onclick()">
  <INPUT ID=btnAmerican league=American value=American type=button>
  <INPUT ID=btnNational league=National value=National type=button>
</DIV>
<SCRIPT LANGUAGE=JavaScript>
function divBtn_onclick() {
  xmlData.src = "getallstardata.asp?league=" + event.srcElement.league;
}
</SCRIPT>
```

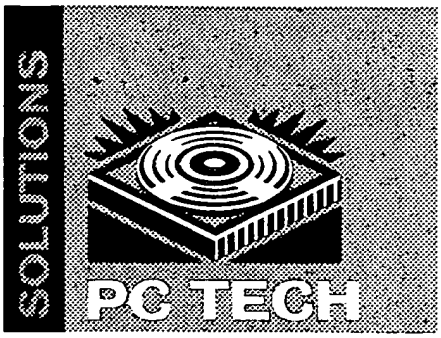
5 This is an XML-based remote procedure call. Depending on which button the user clicks, the divBtn_onclick() function will pass either "American" or "National" as a parameter to getallstardata.asp.

```
<<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="transform.xsl"?>
<allstarstarters>
<allstar>
<ID>1</ID>
  <Player>Ivan Rodriguez</Player>
  <Position>C</Position>
  <Team>Texas</Team>
  <Votes>3012549</Votes>
  <Nice>True</Nice>
  <League>A</League>
</allstar>
<allstar>
  <ID>2</ID>
  <Player>Jim Thome</Player>
  <Position>1B</Position>
  <Team>Cleveland</Team>
  <Votes>1193823</Votes>
  <Nice>True</Nice>
  <League>A</League>
 </allstar>
</allstarstarters>
```

4 Fragment of an XML document that contains a list of all-star major-league baseball players.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="uri:xsl">
  <xsl:template>
    <TD><xsl:value-of /></TD>
  </xsl:template>
<xsl:template match="/">
    <TABLE>
    <TBODY>
    <xsl:for-each select="allstarstarters/allstar">
      <TR>
      <xsl:apply-templates/>
      </TR>
    </xsl:for-each>
    </TBODY>
    </TABLE>
  </xsl:template>
</xsl:stylesheet>
```

6 This XSL stylesheet transforms the XML data in figure 5 into an HTML table.



star baseball player. When the document is read by the XML parser, the parser converts the document from the text-based format you see in Figure 5 to an in-memory object tree. In this way the process is identical to that used by HTML parsers to read an HTML document.

IE5's XML parser lets an application request additional data from a Web server *without discarding the current Web page*. Why is this so important? Today, additional data, embedded in a fresh HTML page that has been dynamically generated by scripting code on the server, is transmitted to the browser. Not only is a whole lot of redundant formatting information retrieved but, even more important, any variables storing the state of the original page must be preserved and transferred to the new page.

Since you can request additional data from a Web server without discarding the current page, you can call server-side functions from within an existing DHTML page. Take a look at the simple DHTML code in Figure 5. Let's assume a user interested in baseball all-stars clicks on the *American League* button. The button's `onclick` event handler then fires and the XML data island's `SRC` attribute is modified. (An *XML data island* is an HTML element that contains the actual XML for use in the HTML document, or contains a reference to a URL that contains the XML.) IE5 then issues an HTTP `GET` request, which is sent to an Active Server Page on the Web server, retrieving an XML document for the American League's all-stars. At this point, the in-memory representation of its data can be manipulated by scripting code using the `xmlData` handle, which was assigned using the ID attribute of the XML data island.

To display the data, you need to convert the XML into HTML, and IE5 supplies the means with the Extensible Stylesheet Language (XSL). XSL lets you write a concise set of rules in XML format that tells the XSL processor how to convert an XML source tree into an XML output tree. In most cases, the XML that is generated is actually HTML that is then rendered for display by the Web browser.

Figure 6 shows a simple XSL file that transforms the XML document shown in Figure 4 into an HTML document. Like all XSL files, this one contains a root template, the `<xsl:template match="/">` instruction. If you look at our template, you will see that we're laying out the skeleton of a document that contains an HTML `<TABLE>` element.

What makes the rest of the stylesheet work is a set of templates that together generate the HTML document. Within the root template, the individual `<allstar>` elements in the source XML file are examined one at a time using the `<xsl:for-each select="allstarstarters/allstar">` instruction. The `<xsl:for-each>` statement generates the HTML table row `<TR>` instruction, and contains an `<xsl:apply-templates/>` statement.

When the XSL processor encounters `<xsl:apply-templates/>`, it begins looking for matches among the other templates defined in the stylesheet. A default template (one that lacks a match attribute) is defined at the top of the stylesheet. It generates the individual cells using the `<TD>` HTML tag and uses the `<xsl:value-of/>` instruction to copy the contents of the currently matched XML element to the output stream. In this case, the contents are copied into the `<TD>` element.

As you can see, XSL lets you write extremely compact rules for transforming XML input trees into XML output trees, and it lets you deal with regular documents, such as the all-star XML document presented here, as well as irregular documents that do not have such well-defined structures. Be sure to work through the excellent XML tutorial on Microsoft's Web site (*msdn.microsoft.com/xml*) to familiarize yourself with the power of XSL stylesheets and the features of the XML parser in IE5.

WRAPPING UP

The major new features IE5 brings to application developers are XML integration and new DHTML capabilities. With the addition of XML data islands and support for XSL processing on the client, IE5 lets you write code that can call out from a Web page to request additional data. DHTML behaviors let Web developers reap the benefits of component-based programming in their applications. They can reuse components written by other developers within their own Web pages and define the behavior of elements across a family of related Web pages using Cascading Style Sheets. IE5 offers many other new developer-oriented features, such as client-side persistence, dynamic properties, and drag-and-drop support. For a detailed look, visit Microsoft's developer Web site, at *msdn.microsoft.com*. PC

John Lam researches Web applications development issues and authored the Essential Web Applications course for DevelopMentor (www.develop.com). You can write to him at jlam@iunknown.com.